X1E Architecture
An Overview of Phoenix

NCCS USERS MEETING

Trey White
Jeff Kuehn
March 27, 2007

Acknowledgements

- **Mike Bast, HPC Operations**
- **Mark Fahey, Scientific Computing**
- **Richard Mills, Scientific Computing**
- **Bill Renaud, User Assistance and Outreach**
- **DOE Office of Advanced Scientific Computing Research**

Outline

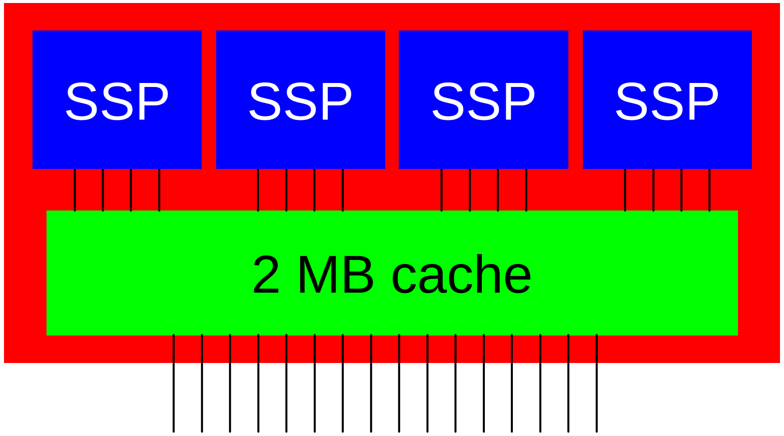
- X1E architecture
- Using Phoenix
- Tuning is required

Phoenix

- **Cray X1E**
- **1024 Multi-Streaming Processors (MSPs)**
  - 18 GF per MSP (and even faster in single precision)
  - 18 TF peak
- **2 TB globally addressable memory**
  - 8 GB per uniform-shared-memory node
  - 2 GB per MSP
- **Powerful interconnect**
  - Enhanced 3D torus
  - Over 10 GB/s measured MPI bandwidth between MSPs

MSP versus SSP

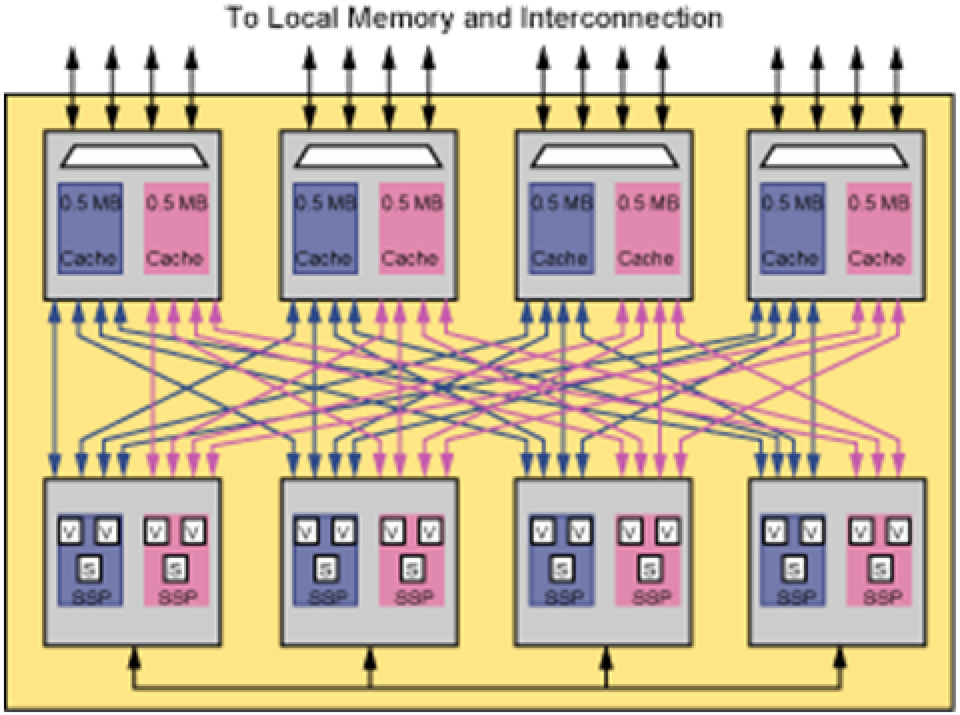
- **Each MSP has four Single-Streaming Processors (SSPs)**
- **Which is “the” processor?**
- **MSP?**
  – 8-pipe vector processor
  – One MPI task
  – Automatic multi-streaming by compiler
  – 2 MB shared cache
  – Most-common mode for real applications



- **SSP?**
  – 2-pipe vector processor
  – Can be an independent MPI process (or OpenMP thread)

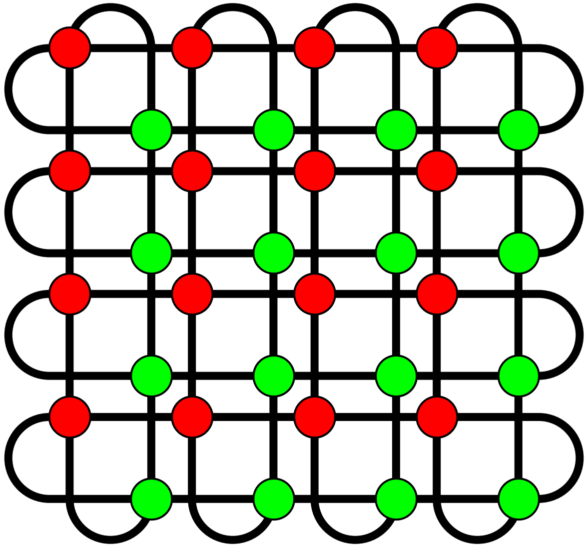
MCMs, nodes, and modules! Oh my!

- **Node = 4 MSPs with uniform shared memory**
- **MCM = Multi-Chip Module = 2 MSPs in different nodes**
- **Module = 1 physical board = 2 nodes**
  - Nodes are interleaved in hardware
  - Separate memory (still globally addressable)
  - Shared memory bandwidth
  - Shared interconnect bandwidth



Interconnect

- **Enhanced 3D torus**
  – Fully connected in one dimension
- **High bandwidth**
  – 10.8 GB/s measured MPI point-to-point
  – Takes four MSPs to saturate module bandwidth
- **Globally addressable memory**
  – Load/store memory on any node
- **Remote address translation**
  – On memory's node, not at processor
  – Avoids TLB misses
  – Requires contiguous processors (default)
  – This is why jobs migrate
- **Cache coherent**
  – Only cache local memory



Many levels of parallelism

- **Vectorization within SSP**
- **Multistreaming within MSP**
- **OpenMP within node (not recommended)**
- **Between nodes (or processors)**
  - MPI-1 two-sided message passing
  - MPI-2 one-sided communication
  - SHMEM one-sided communication
  - Co-Array Fortran remote memory
  - Direct load/store using pointers

X1E strengths

- **Fast vector processors**
  - 18 GF double-precision peak (MSP), 15.3 GF DGEMM
  - Double-rate single precision
- **High memory bandwidth (local and remote)**
  - Stream triad of 32.7 GB/s on one MSP
  - Stream triad of 15.4 GB/s/MSP fully loaded
  - 10.8 GB/s MPI ping pong
  - 108 GB/s Parallel Transpose
  - Good at stride-1, strided, and random access
- **Latency tolerance**
  - Vectorization hides (global and local) memory latency

X1E weaknesses

- **Limited memory per MSP**
- **Very slow scalar processor**
  - 565 MHz
  - 2-way superscalar
  - Simple design (compared to Opteron)
- **Tuning is required**

Outline

- X1E architecture
- Using Phoenix
- Tuning is required

Using Phoenix

- http://nccs.gov
  - → Resources
    - → Cray X1E Phoenix

Login to Robin1

- **4-processor Opteron system with 32 GB of memory**
  - "Robin" will point to Robin1 in a week or so
- **Cross compile for Phoenix**
  - Up to 30x faster than on Phoenix
- **Phoenix "`/tmp/work`" mounted over NFS**
- **Most Phoenix "`man`" pages**
- **Submit and monitor jobs**
  - All PBS and Moab commands
  - Even "`qsub -I`"; shell runs remotely on Phoenix
  - Moab command are slow; Phoenix is the server
- **Typical Linux editors and tools**
  - Emacs, Subversion, *etc.*
  - See "`module avail`"
- **Software auto-configuration can be tricky**

Compiling

- **Avoid "`#ifdef CRAY`"**
  - Cray X1E too different from past Cray's (more like other vendors')
  - Default type sizes are not all 64 bits
- **Use default optimization**
  - Don't try to fix performance problems with higher optimization
- **Always generate loopmarks ("`-rm`", "`-h list=a`")**
- **Often generate instrumented executables ("`pat_build`")**
- **Try newer (or older) compilers with "`module swap`"**
  - `module avail PrgEnv`
  - `module swap PrgEnv PrgEnv.5509`
- **"`-O/h gen_private_callee`" to generate procedure interfaces for calling within CSD streams**
- **"`-Z`" for Co-Array Fortran ("`-h upc`" for UPC)**
- **"`-O/h command`" for serial tools on Phoenix OS nodes**
  - Make sure "`configure`" uses this (when using Phoenix directly)

Batch jobs

- **Scheduling policy not changing (unlike Jaguar)**
- **Always specify requirements in MSPs**
  - "`-l mppe=`*N*"
  - For SSP jobs, divide SSP count by four
  - Jobs using more than one node (4 MSPs) must request an integer multiple of 8 MSPs
    - To line up on module boundaries for remote address translation
- **For more memory**
  - Tell batch system using MSP request (not memory request)
  - Memory/(1.7 GB) = number of MSPs to request
  - Tell "`aprun`" memory requirement using "`-m`"
  - May need to set environment variables
  - See "`man 7 memory`" on Phoenix

Debugging

- **Avoid "`-g`"**
  - Horrible performance
  - Bugs often go into hiding
  - All levels of "`-G`" affect optimization
- **Always set "`TRACEBK`" environment variable**
  - `setenv TRACEBK 30`
  - `export TRACEBK=30`
- **Turn on core files: "`aprun -c core=unlimited`"**
  - Only in "`/tmp/work`"!
- **View core files with "`totalview`" or "`totalviewcli`"**
- **Check traceback for hints on where to look**
  - This says to look at core file #299:

```
Traceback for process 64311(ssp mode) apid 64184.229 on node 7
```

- **See online docs to try interactive debugging**

Outline

- **X1E architecture**
- **Using Phoenix**
- **Tuning is required**

Tuning is required

- **Tuning priorities**
- **Vectorization**
- **Multistreaming**
- **Communication**
- **OpenMP?**
- **Tuning strategy**

Tuning priorities

- **Vectorization (10x)**
- **Multistreaming (4x)**
- **Low-latency communication (2x)**
- **Register blocking (<2x)**
- **Cache blocking?**

Vectorization

- **One vector instruction = many loop iterations**
- **Needs enough loop iterations**
  - 64 (multistreamed) or 256 on X1E
  - Fewer iterations = lower efficiency
- **No procedure calls**
- **No loop-carried data dependencies**
  - Some exceptions (reductions)

Vectorization: What the compiler can do

- **Array notation**
- **Scalar temporary variables**
- **Re-arrange loop nests**
- **Reductions, (un)pack, scatter/gather**
- **Fuse loops and array statements**
- **Inline procedures (one level down)**
- **“`if`” statements within loops**
  - Vector masks, some loss of efficiency

Vectorization: What compilers can't do

- **Make short vector loops efficient**
- **Make stride-1 (or -0) scatter/gather efficient**
- **Know that index arrays don't repeat**
  - ```
    do j = 1, n
       x(i(j)) = x(i(j)) + …
    ```
- **Effectively inline many levels down**
    ```

Vectorization: How you can help

- **Assert that a loop is concurrent (index arrays don't repeat)**
  - `!dir$ concurrent`
  - `#pragma _CRI concurrent`
- **Assert that an index array is a permutation**
  - `!dir$ permutation(i)`
- **Change array temporaries to scalar**
  - Can remove dependencies
- **Break up the big outer loop**
  - To move it inside multiple inner loops
- **Move loops inside procedure calls**
- **Move I/O outside of compute loops**

Vectorization: Loopmark listings

- **What vectorized, what didn't, and why?**

```
679.                           ndayc = 0
680.   Vs------------<         do i=1,ncol
681.   Vs                         if (coszrs(i) > 0.0_r8) then
682.   Vs                            ndayc = ndayc + 1
683.   Vs                            idayc(ndayc) = i
684.   Vs                         end if
685.   Vs------------>         end do
```

```
ftn-6205 f90: VECTOR File = radcswmx.F90, Line = 680
  A loop starting at line 680 was vectorized with a single vector iteration.
```

Beware of partial vectorization

```
6.  Vp----< DO i = 1,n
7.  VP r-<>         e(ix1(i)) = e(ix1(i)) - a(i)
8.  VP----> END DO

f90-6371 f90: VECTOR File = gs-2.f, Line = 6
  A vectorized loop contains potential conflicts due to indirect
  addressing at line 7, causing less efficient code to be
  generated.

f90-6204 f90: VECTOR File = gs-2.f, Line = 6
  A loop starting at line 6 was vectorized.
```

Fix with directives

```
6.          !dir$ concurrent
7.  MV--<         DO i = 1, n
8.  MV              e(ix1(i)) = e(ix1(i)) - a(i)
9.  MV-->         END DO
```

```
f90-6203 f90: VECTOR File = gs-2.f, Line = 7
  A loop starting at line 7 was vectorized because an IVDEP
  or CONCURRENT compiler directive was specified.

f90-6203 f90: STREAM File = gs-2.f, Line = 7
  A loop starting at line 7 was streamed because an IVDEP
  or CONCURRENT compiler directive was specified.
```

Declaring `ix1` *as a* ***permutation*** *may be even better*

Multistreaming

Compiler can multistream:

- **Most vectorizable loops**
- **Most array syntax**
- **Nested loops with no dependencies**
- **Loop nests for vectorization within multistreaming**
- **Short loops**

Compiler can't:

- **Multistream loops with:**
  - Procedure calls
  - Dependencies
- **Always choose the right loop to vectorize versus multistream**

Multistreaming: How you can help

- **Directives, directives, directives**
  - `!dir$ concurrent`
  - `!dir$ preferstream`
  - `!dir$ prefervector`
  - `!dir$ ssp_private`
    (procedure calls)
- **Cray Streaming Directives (CSDs)**
  - Much like OpenMP

I/O inside a loop

```
6.   1--< do i = 1, nx
 7.   1       c(i) = a(i) * b(i)
 8.   1       write(8,'(1x,f12.4)') c(i)
 9.   1--> end do

ftn-6286 ftn: VECTOR File = io1.ftn, Line = 6
  A loop starting at line 6 was not vectorized because it contains
  input/output operations at line 8.

ftn-6709 ftn: STREAM File = io1.ftn, Line = 6
  A loop starting at line 6 was not multi-streamed because it contains
  input/output operations.
```

Fixed

```
 7.   MVr--< do i = 1, nx
 8.   MVr          c(i) = a(i) * b(i)
 9.   MVr--> end do
10.
11.              write(8,'(1x,f12.4)') (c(i),i=1,nx)

ftn-6005 ftn: SCALAR File = io2.ftn, Line = 7
  A loop starting at line 7 was unrolled 2 times.

ftn-6204 ftn: VECTOR File = io2.ftn, Line = 7
  A loop starting at line 7 was vectorized.

ftn-6601 ftn: STREAM File = io2.ftn, Line = 7
  A loop starting at line 7 was multi-streamed.
```

Communication

- **Use one-sided communication for latency-sensitive operations**
- **MPI-2 library**
  - Complicated interface
  - No guaranteed progress without synchronization
- **SHMEM library**
  - Vendor specific
- **Co-Array Fortran**
  - Lowest latency
  - Currently vendor specific
  - Part of next Fortran language standard
- **Intermix with each other and MPI-1**

OpenMP?

- **If OpenMP used for different parallelism than MPI**
  - Probably the same parallelism as for vectorization and multistreaming
  - Typically not enough parallel work for all three
  - OpenMP is least efficient of the three
- **If OpenMP used for same parallelism as MPI**
  - Useful for reducing message volume and aggregating messages
  - But one MSP can't saturate the network
  - Little reason to aggregate
- **Don't bother with OpenMP on Phoenix**

Tuning strategy

- **Functional port**
- **Iterate**
  - Loopmark and profile
  - Vectorize and multistream
- **Tune communication**

Profiling

- **Instrument executable with “`pat_build`”**
- **Run and generate performance report**
  - Together: “`pat_run`”
  - Separately: “`aprun`” followed by offline “`pat_report`” on resulting “`.xf`” file
- **Use report to locate bottlenecks, then use loopmark listings to diagnose problems and solutions**
  - Use call-tree reports to find which calls were expensive
  - Apprentice2 tool provides graphical browsing
- **For aggregate hardware-counter statistics, use “`pat_hwpc`”**

More information

- **This information and more at:**
  http://nccs.gov → Resources → Cray X1E Phoenix
  http://info.nccs.gov → Cray X1E Phoenix
  http://info.nccs.gov/resources/phoenix
- **Cray documentation at http://docs.cray.com**
  - *Cray X1 Series System Overview*
  - Cray Fortran, C/C++ reference manuals
  - *Migrating Applications to the Cray X1 Series Systems*
  - *Optimizing Applications on Cray Series Systems*
  - Excellent search capability
- **E-mail us: help@nccs.gov**